2019

A fully trainable network with RNN-based pooling

Shuai Li
University of Wollongong, University of Electronic Science and Technology of China, sl669@uowmail.edu.au

Wanqing Li
University of Wollongong, wanqing@uow.edu.au

Christopher David Cook
University of Wollongong, chris_cook@uow.edu.au

Ce Zhu
University of Electronic Science and Technology of China

Yanbo Gao
University of Electronic Science and Technology of China

A fully trainable network with RNN-based pooling

Abstract

Pooling is an important component in convolutional neural networks (CNNs) for aggregating features and reducing computational burden. Compared with other components such as convolutional layers and fully connected layers which are completely learned from data, the pooling component is still handcrafted such as max pooling and average pooling. This paper proposes a learnable pooling function using recurrent neural networks (RNN) so that the pooling can be fully adapted to data and other components of the network, leading to an improved performance. Such a network with learnable pooling function is referred to as a fully trainable network (FTN). Experimental results demonstrate that the proposed RNN based pooling can well approximate the existing pooling functions with just one neuron, thus making it appropriate to be used as pooling function in a network with its rich representation capability. Furthermore, experiments have shown that the proposed FTN can achieve better performance than the existing pooling methods under similar network architectures for image classification.

Disciplines

Engineering | Science and Technology Studies

Publication Details

Li, S., Li, W., Cook, C., Zhu, C. & Gao, Y. (2019). A fully trainable network with RNN-based pooling. Neurocomputing, 338 72-82.

A Fully Trainable Network with RNN-based Pooling

Shuai Li[a,*], Wanqing Li[a], Chris Cook[a], Ce Zhu[b], Yanbo Gao[b]

[a]*School of Computing and Information Technology, University of Wollongong, NSW 2522, Australia*
[b]*School of Electronic Engineering, University of Electronic Science and Technology of China, Chengdu 611731, China*

Abstract

Pooling is an important component in convolutional neural networks (CNNs) for aggregating features and reducing computational burden. Compared with other components such as convolutional layers and fully connected layers which are completely learned from data, the pooling component is still handcrafted such as max pooling and average pooling. This paper proposes a learnable pooling function using recurrent neural networks (RNN) so that the pooling can be fully adapted to data and other components of the network, leading to an improved performance. Such a network with learnable pooling function is referred to as a fully trainable network (FTN). Experimental results demonstrate that the proposed RNN based pooling can well approximate the existing pooling functions with just one neuron, thus making it appropriate to be used as pooling function in a network with its rich representation capability. Furthermore, experiments have shown that the proposed FTN can achieve better performance than the existing pooling methods under similar network architectures for image classification.

Keywords: Pooling, recurrent neural network, convolutional neural network, deep learning

1. Introduction

Convolutional neural networks (CNNs) have recently achieved the state-of-the-art performance in many image analysis tasks [1, 2, 3, 4]. It has also been shown to be very effective in extracting features for action recognition and other tasks involving sequential data [5, 6, 7, 8]. In most (if not all) of the existing CNN architectures such as the "AlexNet"[1], "VGGNet"[2] and "InceptionNet"[3], pooling is an important component for aggregating local features and reducing computational burden. In some networks [9], convolution with strides (larger than 1) is used to reduce the dimension of features to achieve a function similar to pooling. Noticeably, pooling is the only component in a typical CNN architecture (without considering normalization layers which are mostly for fast training and convergence) that is completely engineered with prior knowledge (such as max pooling and average pooling) instead of learning from data. Since the power of CNNs comes from their ability to adapt to the data through learning, the natural question to ask is "could pooling be learned in a similar way as other components from data?". To answer this question, this paper proposes a learnable pooling function based on recurrent neural units. Together with the convolutional layers and fully connected layers in CNNs, such a learnable pooling leads to a fully trainable network (FTN). Compared with the traditional CNNs, it has the benefit of being fully adapted to data and task. Experimental results have demonstrated that FTN can improve the performance over CNNs with similar network architectures but fixed pooling methods, especially on small networks.

The main contributions of this paper are summarized as follows.

- We propose a RNN based pooling method which is learnable and can be trained with data and task. Together with the CNNs, this leads to a fully trainable network (FTN), which can be trained end-to-end.
- Since RNNs are universal approximators, the proposed RNN based pooling can be trained to be optimal for FTN. We have shown that one RNN unit is able to approximate the existing average and max pooling with a high accuracy. Therefore, it can be easily used to replace the existing pooling operations in the existing models and be further fine-tuned, in addition to being trained end-to-end.

*Corresponding author

Email addresses: shuailichn@gmail.com (Shuai Li), wanqing@uow.edu.au (Wanqing Li), ccook@uow.edu.au (Chris Cook), eczhu@uestc.edu.cn (Ce Zhu), gyb_chn@163.com (Yanbo Gao)

- With the proposed RNN based pooling, FTN has shown to be able to improve the performance over CNN in the classification tasks.

The rest of the paper is organized as follows. The related work is discussed in Section 2. Section 3 presents the proposed method and experimental results are shown in Section 4. Conclusion is drawn in Section 5.

2. Related work

2.1. Pooling

Motivated from biology [10] where responses of simple cells are fed into complex cell through some pooling operations, the spatial pooling approach has been found very useful in many computer vision tasks. Up to now, the most commonly used pooling methods are still the max pooling and average pooling. The max pooling selects the most salient feature in a pooling region while the average pooling treats the features in a pooling region equally. However, the features in a local pooling region may be heterogeneous, leading to a loss of information on weak features through max pooling and loss of discriminative information through average pooling [11]. It has been shown in the research that such pooling methods cannot achieve the optimal performance due to this information loss [11]. A theoretical analysis of max pooling and average pooling for classification is provided in [12] based on the i.i.d. Bernoulli distribution assumption for binary features and the exponential distribution for continuous features. It shows that the pooling cardinality and sparsity of the features affect its classification performance and the performance highly depends on the distribution of the features which is hard to estimate.

In addition to the max pooling and average pooling, there are some other pooling methods reported in the literature. In [13], a protected pooling method was proposed where a concave function is used to combine the features. The concave function is designed to protect weak codes in order to preserve details. In [14], a stochastic pooling method was proposed where a multinomial distribution formed from the activation values is used. In this way, the location with large values are picked as output more frequently than others. Similarly in [15], a rank based pooling was proposed to emphasize information with high rank over others. In [16], spectral pooling was proposed which preforms dimensionality reduction by truncating the representation in the frequency domain. These methods are all heavily engineered to certain functions irrespectively of datasets, tasks and architectures of the networks. On the other hand, the convolution with sliding strides larger than one pixel [9] can be regarded as an extra convolutional layer with a pooling operation which selects the value of a fixed location. The learning pooling in [17] further simplifies the convolutional operation with independent linear operation on each channel.

There are also methods proposed to combine different pooling functions. In [18], a mixed pooling method was proposed where max pooling and average pooling are randomly selected with a stochastic procedure. Similarly, in [19], the max pooling and average pooling are combined in a tree structure. In [20] and [21], a geometric l_p-norm pooling was proposed to generalize the max pooling and average pooling, which can be represented as $(\sum_{i=1}^{N} |x_{I_i}|^p)^{1/p}$. When $p = 1$, l_p-norm pooling reduces to average pooling and when $p = \infty$, l_p-norm pooling reduces to max pooling.

Pooling is a process that maps the $N \times N$ to 1 where $N \times N$ represents the size of the local pooling region. The existing pooling methods certainly lose information in this mapping process. Compared to the other layers in CNNs such as the convolutional layers and the fully connected layers, the predefined pooling may not be the optimal pooling function for the given data and task. Moreover, different pooling methods are used in different CNN architectures. Even in one CNN architecture such as the "InceptionNet"[3], different pooling methods are used. While it is difficult to select an appropriate pooling strategy for a better performance, it is also hard to explain how and why one pooling strategy works better than others. Therefore, a flexible pooling function that can be learned from data for each pooling layer of a network architecture is highly desired.

2.2. Recurrent neural networks

Recurrent neural networks (RNNs) [22] and their variants such as long short-term memory unit (LSTM) [23], gated recurrent unit (GRU) [24], and independently recurrent neuron network (IndRNN) [25] have been shown to be capable of aggregating sequential information and recognizing patterns in a sequence[7]. A common framework of employing RNNs is the encoder-decoder model [26]. The encoder RNNs map an input sequence into a fixed length representation and then the decoder RNNs, based on the representation, are used to perform different tasks such as prediction. In addition to the encoder-decoder model, RNNs have also been used in scene analysis [27], image generation [28], action recognition [29], etc. In [27], a two-dimensional RNN network is used to model the spatial dependency in an image for scene labeling. In [30], a Directed Acyclic Graph - Recurrent Neural Networks

(DAG-RNN) was proposed to perform context aggregation for scene segmentation. The image is first decomposed into a set of directed cyclic graphs and then each of them is processed by DAG-RNN independently. This process allows the local information to be routed to anywhere in the image, which greatly enhances the exploration of context information. In [31], a long short-term memorized context fusion (LSTM-CF) was proposed for RGB-D scene labeling where LSTM is used to explore the context in both RGB and depth images. In [32], a convolutional recurrent neural network was proposed for image-based sequence recognition. RNN is employed on top of the convolutional layers to capture contextual information. In [33], a deep recurrent attention writer (DRAW) neural network was proposed for iterative generation of complex images where RNNs are used to process the image and iteratively provide attention regions for reading and writing. In [29], RNN was used for skeleton based action recognition where a hierarchical architecture is constructed to better explore the relationship of different joints in different parts of the body. In [34], the RNN model was further combined with a global attention layer to selectively focus on different joints for action recognition. In summary, from the perspective of sequence modelling, RNNs have been more popular than CNNs. Although there are some works using CNNs for sequence modelling such as the convolutional sequence to sequence model [35], the convolution kernel in the time dimension is generally small such as 3, and it requires a deep network to process a sequence. On the contrary, RNNs generally only require one or two layers to effectively process a sequence.

Since the objective of pooling is to aggregate features of a local region, it is possible to consider RNN as a pooling function, especially considering that RNNs are universal approximators (Turing-Complete [36]) and can be trained based on the data together with other parameters in the network. However, in practice, if the number of neurons required to approximate the pooling function is too large, it will make the training process inefficient and thus affect the performance of the whole neural network adversely. In this paper, we show that one RNN unit with modified activation functions is able to well approximate the existing max and average pooling methods. Therefore, better performance can be expected by incorporating the proposed RNN based pooling into CNNs.

3. The proposed fully trainable network

3.1. Overview

The basic component of a CNN is a stack of convolutional layers (usually more than 2) followed by a pooling layer as shown in Fig. 1(a). The convolutional layer can be of many forms such as the traditional convolution structure [2], inception structure [3] and the residual structure [37]. Normalization layers [38] may be used after or before each convolutional layer which is not considered here. The pooling layer is often a max pooling, average pooling or a pooling function as discussed above. Instead of using a pooling layer, a convolutional layer with stride larger than 1 can be used [9] as shown in Fig. 1(b) to reduce the dimension of the output features. In the proposed FTN, the basic component is a stack of convolutional layers followed by a recurrent layer as shown in Fig. 1(c). Specifically, the features in each pooling region are scanned into a sequence as input to the recurrent layer. There are many ways to perform the scan. The output of the recurrent layer at the last time stamp is the aggregated feature of the local pooling region, and so is treated as the pooled value. It has been empirically shown that the performance of the FTN is insensitive to the scanning order. Thus simple horizontal scanning is adopted in this paper. In addition to reducing the dimension of the features, the recurrent layer also intends to capture the pattern of the features in a local region. A FTN is constructed by stacking such components.

Notice that in general any type of RNN can be used in the FTN for pooling. This paper adopts the commonly used LSTM unit. In the following, FTNs are explained in detail with respect to the extension of an LSTM unit for pooling, and the FTN architectures, respectively.

3.2. Extension of an LSTM unit for pooling

In the study of CNNs, a general consensus is that for deep networks non-saturated activation functions such as rectified linear units (ReLU) are easier to train than the saturated activation functions such as logistic and hyperbolic tangent functions. In this paper, it is proposed to extend a conventional LSTM unit with non-saturated activation functions to preform pooling. Such an extension facilitates the training of the LSTM in a consistent way with other layers in a FTN.

The key component in LSTM [23] is a constant error carousel (CEC) which enforces a constant error flow over time steps. Fig. 2 illustrates an LSTM without considering peephole connections. In addition to the CEC, LSTM contains three gates (input gate, forget gate and output gate), and two modulations (input modulation and output modulation). The gates are controlled by the current input and the recurrent input. The activation function for the gates are usually the sigmoid function (σ). The activation functions (ψ) used in input and output modulations are

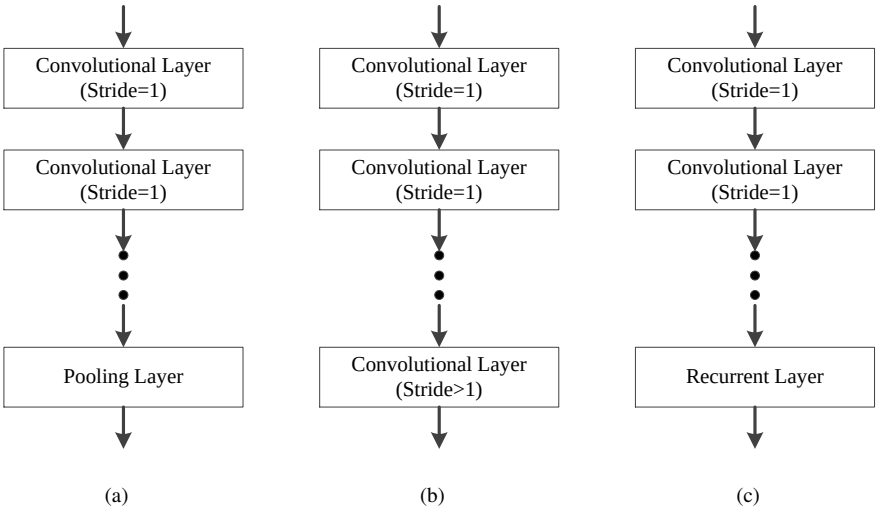


Figure 1: Key components in a CNN with (a) a pooling layer, (b) a convolutional layer with stride larger than 1, (c) a recurrent layer.

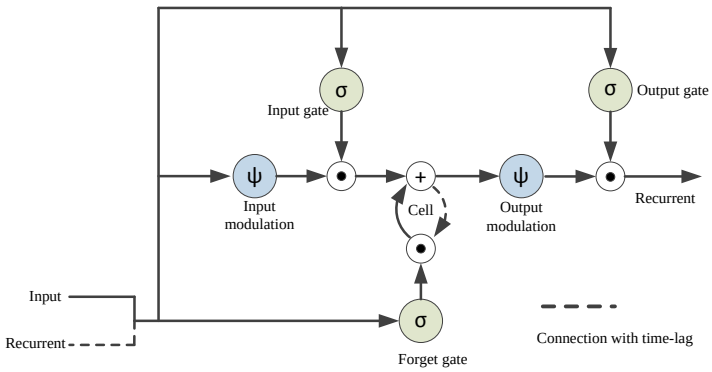


Figure 2: Illustration of a LSTM unit.

usually the hyperbolic tangent function ($tanh$). For input $\mathbf{x}$ at each time step t, the LSTM updates its states as follows:

$$
\begin{aligned}
\mathbf{i}^t &= \sigma(\mathbf{W}_i\mathbf{x}^t + \mathbf{R}_i\mathbf{h}^{t-1} + \mathbf{b}_i) \\
\mathbf{f}^t &= \sigma(\mathbf{W}_f\mathbf{x}^t + \mathbf{R}_f\mathbf{h}^{t-1} + \mathbf{b}_f) \\
\mathbf{o}^t &= \sigma(\mathbf{W}_o\mathbf{x}^t + \mathbf{R}_o\mathbf{h}^{t-1} + \mathbf{b}_o) \\
\mathbf{g}^t &= \psi(\mathbf{W}_g\mathbf{x}^t + \mathbf{R}_g\mathbf{h}^{t-1} + \mathbf{b}_g) \\
\mathbf{c}^t &= \mathbf{i}^t \odot \mathbf{g}^t + \mathbf{f}^t \odot \mathbf{c}^{t-1} \\
\mathbf{h}^t &= \mathbf{o}^t \odot \psi(\mathbf{c}^t)
\end{aligned} \tag{1}
$$

where $\mathbf{x}^t \in \mathbb{R}^M$, $\mathbf{h}^{t-1} \in \mathbb{R}^N$ and M, N represent the dimension of the input feature at time step t and the number of the neurons in LSTM, respectively. $\mathbf{i}^t$, $\mathbf{f}^t$ and $\mathbf{o}^t$ are the outputs of the input gate, forget gate and output gate, respectively. $\mathbf{g}^t$, $\mathbf{c}^t$ and $\mathbf{h}^t$ are the output of the input modulation, the cell state and the output of the LSTM, respectively. $\mathbf{W}_v$, $\mathbf{R}_v$ and $\mathbf{b}_v$ are the weight of the current input, the weight of the recurrent input, and the bias, respectively, for the input gate ($v = i$), forget gate ($v = f$), output gate ($v = o$) and input modulation ($v = g$). $\odot$ represents the point-wise multiplication.

With the hyperbolic tangent function used as the activation function for input and output modulations, the output of the LSTM is constrained to the range of $(-1, 1)$. However, the convolutional layers and fully connected

layers in most CNN architectures employ non-saturated activation functions such as ReLU where their output ranges in $[0, +\infty)$. Therefore, the activation functions of the input and output modulations in a LSTM unit are required to be the same as those used in the convolutional layers.

However, change of the activation functions of the input and output modulations in a LSTM unit from a saturated function to a non-saturated function would usually make the training of the LSTM hard to converge according to [39]. In this paper, one LSTM unit is applied to pool features in a local patch in each channel. Since the processing of all samples in the batch is the same, we take the processing of one data sample as an example to illustrate the LSTM based pooling process. Accordingly, the dimension of input feature from one data sample and the number of units are both 1, i.e., $M = 1$ and $N = 1$. That is, the input, recurrent input, cell state and outputs of all the gates at each time instance and the corresponding weight parameters are all of dimension 1. Let them be noted as x^t, h^t, i^t, f^t, o^t, c^t, w_g, r_g, w_v, r_v, b_v, respectively. The following proposition is used as a regulation in the training of LSTM.

Proposition: $w_g > 0$ is a necessary condition for a LSTM unit with ReLU activation function to converge when processing non-negative input features (x^t).

Proof: (Proof by Contradiction) Let the bias of the input modulation be first ignored and considered later, that is, $g^t = \psi(w_g x^t + r_g h^{t-1})$, where ψ is the ReLU activation function and $x^t \geq 0$. The initial state of the recurrent input (h^0) and cell (c^0) are both set to be 0 which is used in most networks. Assume $w_g \leq 0$. Starting with time instance 1, with $x^1 \geq 0$, the output of the input modulation g^1 is zero. Since the outputs of all the gates including input gate, output gate and forget gate are non-negative, the output of LSTM (h^1) and the cell state (c^1) is 0. Hence, the recurrent input and the cell state for the next time instance 2 remains 0. Together with $x^2 \geq 0$, the output stays 0. It can be deduced that under such circumstances, the output of LSTM stays 0, which cannot be trained to converge. Therefore, the assumption does not hold and the opposite proposition ($w_g > 0$) is true. On the other hand, bias determines the threshold to activate a neuron. For a LSTM unit, a negative bias for the input modulation deactivates the neuron for small inputs, making the neurons incapable of processing small features. Therefore, the bias for the input modulation is suggested to be constrained to be non-negative as well, in order to preserve the unit's ability of dealing with small inputs.

Experimental results have shown that a LSTM unit with ReLU activation function can be trained robustly if this proposition is met.

3.3. FTN architectures

The proposed LSTM based pooling can be integrated with convolutional layers in different ways to create different FTN architectures.

- One FTN architecture is that each local pooling region has its own LSTM to be trained and these LSTM units can work as different pooling functions for different regions. In this FTN, pooling is adaptive to local regions.
- The second FTN architecture has one LSTM per layer that is shared by all local regions in the layer. In this case, one pooling operation is performed on all local regions. However, pooling at different layers can be different depending on the training. For instance, the LSTM in one layer may act like a max pooling and the LSTM in another layer may act like an average pooling or a different function that the LSTM would best approximate for the training data.
- The third FTN architecture is one LSTM unit shared by all pooling layers. This is equivalent to the conventional CNN where either max or average pooling is adopted.

Obviously, the first architecture has the maximum number of parameters to be trained for the pooling and so is not considered further. In the experiments, the second and third FTN architectures are evaluated and compared to illustrate the benefits of the proposed LSTM based pooling over the traditional pooling.

4. Experimental Results

Experiments were first conducted to validate that one LSTM unit can well approximate the max and average pooling functions, thus showing the proposed LSTM based pooling is appropriate to be used as pooling in the network. Next, experiments were devised to illustrate that an adaptive pooling such as the proposed LSTM based pooling can outperform the predefined max or average pooling and analysis on the learned pooling function is provided. Finally, the performance of FTN is verified on the conventional classification tasks by comparing with the existing methods.

Table 1: MAE(10^{-5}) of one LSTM unit with the modified activation function to approximate a max pooling function.

	T1	T2	T3
Pool size 2×2	8.97	8.91	9.19
Pool size 3×3	4.42	4.39	4.40
Pool size 4×4	5.41	5.28	5.32

Table 2: MAE of one hidden layer feedforward neural network to approximate a max pooling function.

Pool size 2×2	Pool size 3×3	Pool size 4×4
38	20	12

4.1. LSTM for average and max pooling

The average pooling is a simple linear function which can be easily approximated by LSTM. However, the max pooling function is a highly non-linear function. In the following, an experiment was devised to show that one LSTM unit is able to approximate the max pooling function to a high degree of accuracy.

Simulation setup: ReLU was assumed as the activation function for the convolutional layers that a LSTM unit would work with. Experiments for other non-saturated activation functions can be conducted in a similar way. For ReLU ($max(0, x)$), the range of the output in theory is $[0, +\infty)$. In experiments of image classification, it is observed that the outputs generally fall in the range $[0, 300]$. Therefore, random numbers in this range were generated as the input to simulate the output of a convolutional layer. Since the pooling sizes used most in CNN are 2×2, 3×3 and 4×4, three sets of experiments were conducted with lengths of the input being 4, 9 and 16, respectively. One LSTM unit with the modified activation function (ReLU here) was used.

Training: The LSTM was trained by minimizing the mean absolute error (MAE) between the output and the max value of the input to approximate the max pooling. Mini-batch gradient descent with Nesterov momentum [40] was used and the batch size was set to 128. The initial learning rate was set to 0.1 and the momentum was set to 0.9. Regularizations such as weight decay and dropout were not used since infinite training examples can be generated. 10^4 batches were considered as an epoch and one epoch was used for validation. The learning rate was decreased by a factor of 10 when the validation accuracy stopped improving. The input weight and bias of the input modulation in the LSTM unit was initialized and regulated as described in subsection 3.2.

Testing: The batch size used for testing is the same as that for training. One epoch (10^4 batches) of data was generated for testing. The performance of the trained network is evaluated on three sets of input data: T1: random numbers in the range $[0, 300]$; T2: 50% of random numbers in the range $[0, 300]$ and the other 50% being 0; T3: 20% of random numbers in the range $[0, 300]$ and the other 80% being 0. The tests were designed to simulate the cases of general patches, relatively sparse patches and highly sparse patches considering that the responses of the convolutional units can be sparse. The performance for different pooling sizes are tabulated in Table 1.

From Table 1, it can be seen that one LSTM unit is able to well approximate the max pooling function as the errors are all smaller than 10^{-4} (which is negligible compared to the data range of $[0, 300]$). It can be also seen that the performance is insensitive to the pooling sizes. On the other hand, it has been shown in the literature that feedforward neural networks can also function as universal approximator provided there are sufficient number of hidden units [41]. Here experiments using the feedforward network with different numbers of hidden units to function as pooling are further conducted. Similarly as the above experiment using LSTM to approximate max pooling, one hidden layer neural networks are also tested to first approximate the widely used max pooling. Various numbers of neurons including 10, 50, 100 and 500 are tested, respectively, where ReLU is used as the activation function. The simulation setup is the same as above where random numbers in the range of $[0, 300]$ are generated. Different pooling sizes including 2×2, 3×3 and 4×4 are tested where the maximal value is used as the target. The learning rate of 0.1, 0.01 and 0.001 are all used and the best result is reported as shown in Table 2. It can be seen that a one hidden layer feedforward neural network cannot approximate max pooling well where the MAE is no less than 12 for all tested pooling sizes even with a relatively large number of neurons. Note that the expectation of the maximum of n random variables $(N_1, N_2, ..., N_n)$ uniformly distributed in the range of $(0, 1)$, i.e., $E(max(N_1, N_2, ..., N_n))$, is $n/(n + 1)$. Therefore, for pooling size 2×2 where $n = 4$, the expectation of the max pooling over $[0, 300]$ is 240. Similarly, for pooling size 3×3 and 4×4, the expectations are

Table 3: Classification result comparison on CIFAR-10 in terms of test error rate (%) using different sizes of networks.

Network	Max pooling	Average pooling	Proposed pooling (shared)	Proposed pooling
Conv_8	57.44	56.18	50.96	50.52
Conv_16	32.75	32.86	28.58	25.72
Conv_32	18.77	20.82	16.11	15.48
Conv_64	13.27	14.75	12.04	11.83

270 and 282, respectively. In our experiments, we found that all the outputs are around the above expectation of the max pooling without capturing the maximal value of each individual input, indicating that a one hidden layer feedforward neural network with a limited number of hidden units cannot simply approximate the max pooling. On the contrary, LSTM with only one unit can approximate max pooling very well where the MAE is smaller than 10^{-4} as shown in Table 1.

In the following, the performance of FTN over CNNs will be illustrated for different datasets and the learned pooling functions will be investigated.

4.2. Analysis of the LSTM based pooling

4.2.1. Performance on different sizes of networks

To illustrate the effectiveness of the proposed LSTM based pooling, experiments were conducted on the popular CIFAR-10 dataset. The dataset was preprocessed in the same way as in [42]. That is, the dataset is preprocessed with global contrast normalization and ZCA whitening, and the images were padded with four zero pixels at borders. While training, 32×32 random crops with random horizontal flipping were used as input.

A CNN, composing of two stacks of 3×3 convolutional layers (2 layers in each stack) with a pooling layer at the end of each stack, 2 fully connected layers, and an additional fully connected layer of 10 units together with a softmax output layer for classification was used. The same number of units, denoted by N, were used in the convolutional layers and the 2 fully connected layers. The corresponding network is denoted as Conv_N. To better demonstrate the effectiveness of the LSTM based pooling, a large local pooling region, namely 4×4 and 8×8 for the first and second pooling layers, respectively, were used. After the pooling layers, the size of the input to the fully connected layers is 1×1, thus the fully connected layers work in the same way as convolutional layers. A leaky ReLU unit with leakiness of 0.3 was used as the activation function for both convolutional layers and fully connected layers, which has been reported [43] to achieve a good performance on classification. Batch normalization [38] was used for convolutional layers and dropout (drooping rate 50%) was applied after each fully connected layer. Total norm constraint on the gradients as in [44] was used to stabilize the training. The initial learning rate was set to 0.01 and decreased by a factor of 10 after 50k and 40k iterations, respectively, and the training ended at the 122K-th iteration. SGD with Nesterov momentum [40] of 0.9 was used for training, and the batch size was 100.

The results are shown in Table 3. The column of "Proposed pooling" and "Proposed pooling (shared)" represent the second and third architectures described in Subsection 3.3, respectively. That is to say, for "proposed pooling (shared)", both pooling layers share one LSTM unit while for "proposed pooling", each pooling layer has one LSTM unit. From the table, three observations can be made:

- The networks with the proposed LSTM pooling always improve the accuracy (lower the error rate) compared with the corresponding CNNs coupled with the traditional max pooling or average pooling function.
- When the network is very small such as Conv_8, the performance improvement due to the LSTM based pooling is significant, up to 7 percentage points. As the network size increases, the improvement decreases. It is conjectured that although a fixed pooling function may not optimally aggregate the local features, extra convolution kernels may compensate this to some extent. Thus with the increase of the convolution units, the gain of using a better pooling function over traditional pooling method drops. However, given the data, if the network is too large, it may become overfitting. Therefore, a better pooling function to aggregate the features while reducing the requirement of units is highly desired.
- The performance of the second architecture of FTN is better than the third, i.e., different LSTM units for different pooling layers improves the performance of FTN. This indicates that the optimal pooling functions for different pooling layers are likely to be different.

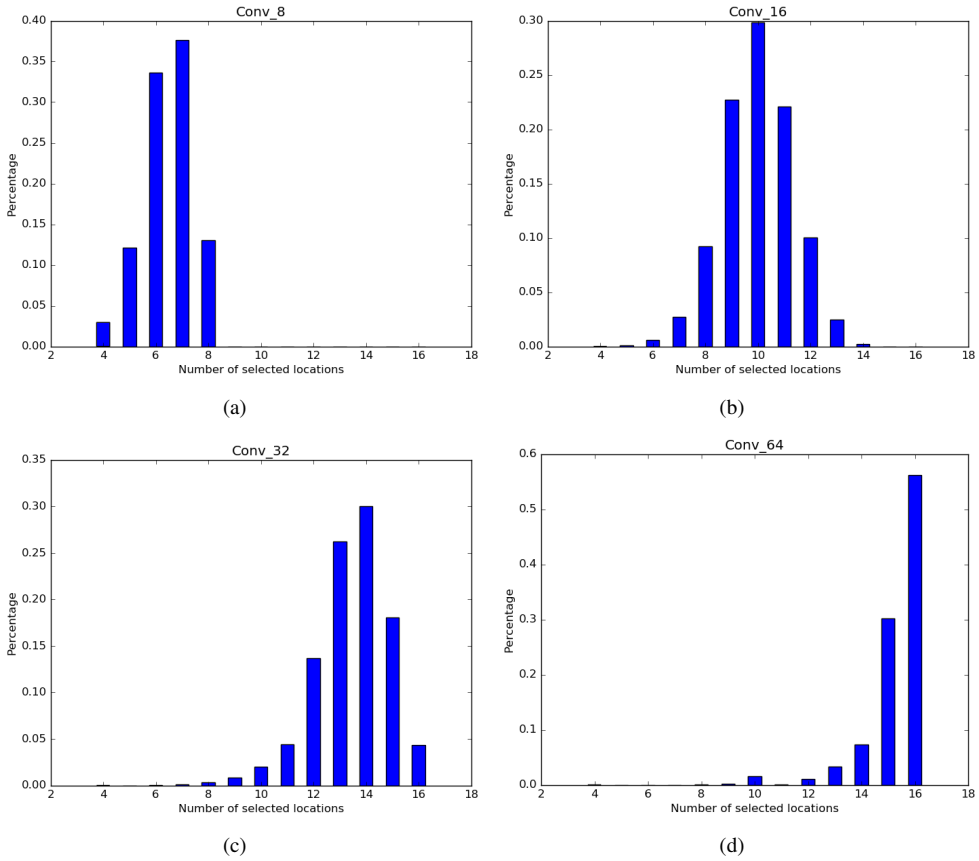
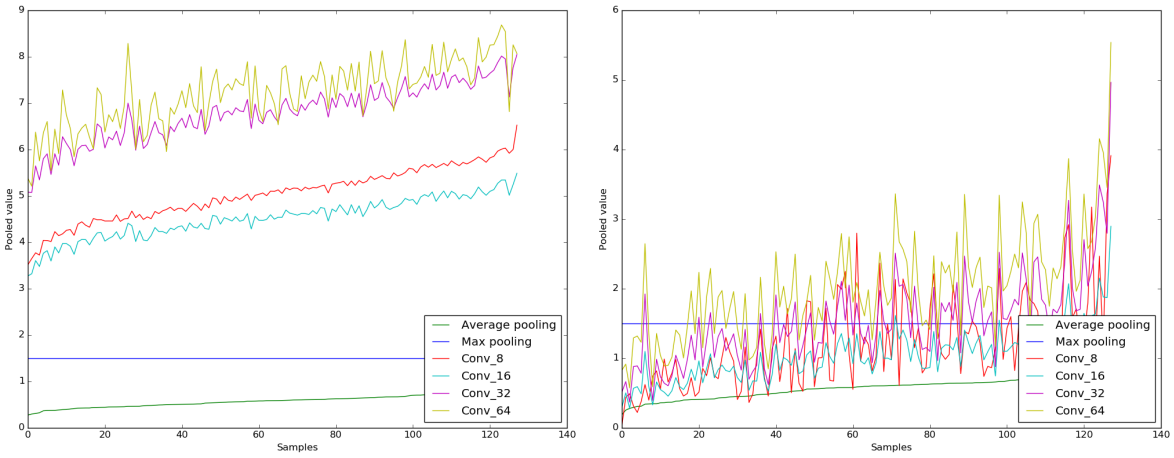


Figure 3: Illustration of the location selection in the max pooling over different sizes of the networks.

4.2.2. Analysis of the learned pooling function

As shown in Table 3 and described above, the performance gap between the proposed pooling and the existing pooling methods becomes smaller with the increase of the number of convolution kernels, especially for the max pooling. Since pooling is a $N \times N$ to 1 mapping process, information of certain locations in the pooling region may be lost in the existing pooling process, leading to a degraded performance of the network. In the following, we first show that networks are trained to preserve information of different locations in a pooling region.

For a CNN, max pooling is used independently for each channel. And for each channel, it selects the value of one location (which is the location with the maximal value) as the output and the information at that location is implicitly carried forward in this channel. That is to say, for the pooling process over multiple channels, information from a number of locations may be selected and preserved by one or more channels. Fig. 3 shows the histogram of the number of locations that have been selected by at least one channel. The output of max pooling for 5000 randomly selected local patches in the dataset are used for different networks. The pooling size is $4 * 4$ in all networks, leading to a local region of 16 locations. For the Conv_8 network, the number of neurons is 8, and thus at most 8 locations can be selected by the max pooling (when locations selected by different channels are all different). Fig. 3(a) shows that generally more than 4 locations have been selected in one or more than one channels, and for some pooling regions, all 8 locations are selected. For the Conv_16 network shown in 3(b), generally more than 6 locations have been selected. For the Conv_64 network shown in 3(b), in over 50% pooling regions, all 16 locations have been selected. It is reasonable to assume that when the number of the neurons is large enough, information from all locations may be implicitly carried forward after the pooling operation. It indicates that networks (convolution kernels) are trained to sample information from all locations. Consequently, with the increase of the number of neurons, the effect of a good pooling operation may be reduced since information from more locations could be sampled through different channels. However, in this case, noise may also be kept by different channels and when the number of neurons is very large, the network becomes overfitting and degrades the performance. On the other hand, since LSTM can aggregate information of a sequence, it can adaptively sample more information from all the pooling locations than the existing pooling functions. Therefore, the performance



(a) (b)

Figure 4: Outputs of the learned pooling function from networks of different sizes in comparison with the max pooling and average pooling. (a) Learned function of the first pooling layer, (b) learned function of the second pooling layer.

of the proposed FTN is significantly better than the traditional CNN when the number of neurons used is small. Moreover, compared to CNN, FTN can lower the requirement of the number of neurons for a better performance in order to avoid being overfitting.

To illustrate the learned LSTM based pooling functions for different pooling layers in the network of different sizes, the output of the pooling function in comparison with the max pooling and average pooling is shown in Fig. 4. For better illustration, random values with a fixed maximum value (1.5) is used as input and the output is rearranged according to the magnitude of the average pooling result. Fig. 4(a) shows the output of the learned pooling function from the first pooling layer. "Conv_N" represents the outputs obtained from the different pooling functions learned from their corresponding "Conv_N" networks, respectively. Note that the mean value of each pooling result can be compensated by the bias of the neurons in the following layer, thus the variation of each pooling result is more meaningful. It can be seen that the learned pooling functions of the first pooling layer work similarly as the average pooling. Especially for the small networks such as "Conv_8" and "Conv_16", the output highly correlates with the output of average pooling and the variation is relatively small. This indicates that average pooling may perform better than max pooling for the first pooling layer of small networks, which agrees with our results shown in Table 3.

For the learned pooling function of the second pooling layer as shown in Fig. 4(b), it can be seen that the variation is very large, i.e., highly sensitive to the different patterns of the inputs. First, compared with the input to the first pooling layer, each input to the second pooling layer corresponds to a larger region of the original image and thus more useful information for the task. Second, it is known that outputs of the higher layers in the network capture high level information, and information at different locations may produce different contexts for the final classification task. For example, the same input with different orders may produce different results. Thus it is very important for the pooling layer to aggregate information while capturing useful patterns. This can be done using our proposed pooling while not possible for the traditional max pooling and average pooling.

By comparing the learned pooling functions of two layers, it can be seen that the optimal pooling functions for different pooling layers are quite different. In the traditional CNNs, max pooling and average pooling are often selected empirically. In the "InceptionNet" [3], both max pooling and averaging pooling are adopted at different layers also empirically. In such a case, the whole network cannot achieve the best performance. On the other hand, our proposed LSTM based pooling is able to be adaptive for each layer to the training data.

4.2.3. Complexity

As mentioned in Subsection 3.1, the proposed LSTM based pooling first transforms the $N \times N$ local region to a sequential input of length $N \times N$. Then it processes this sequential input. For example, for the general 2×2 pooling, LSTM needs to process inputs of 4 time steps. It is known that the update of LSTM at each time step in Eq. (1) can be regarded as convolution with multiple channels. So the complexity of the proposed pooling is similar to performing 4 convolutional layers. To evaluate its complexity, a similar network as the Conv_64 network (except

Table 4: Complexity comparison between CNN and the proposed FTN in terms of time (sec per batch).

	Train with cudnn	Train without cudnn	Test with cudnn	Test without cudnn
CNN	0.013	0.021	0.0027	0.0037
FTN	0.033	0.035	0.0068	0.0076

that the pooling size is set to be 2×2) was used. Batch size was set to be 1 to purely monitor the computation without considering memory issues. The program was implemented based on Theano [45] and Lasagne, and runs on a TITAN X GPU. The time used in the training and testing process is shown in Table 4. Since convolution is heavily optimized in cudnn (the deep neural network library developed in NVIDA CUDA), we show both results obtained with cudnn and without cudnn. It can be seen that the complexity of training the above FTN network is about two-three times of training CNN. This is consistent with our above analysis that the complexity of the proposed LSTM based pooling is similar to performing 4 convolutional layers. Since pooling is only applied a few times depending on the size of the input (around 5 times for input of size 256), the complexity of training FTN is bounded. Compared to the current networks over 100 layers, the increase in training time is acceptable considering the benefit of having a learnable pooling function to improve the performance. Moreover, compared to the image modelling methods that use RNN to process the whole image in a sequential manner, the time increase due to the proposed pooling is relatively very small. It is worth noting that since the proposed pooling is learned from data for a network, it can be used as a tool to develop new pooling functions for different applications.

4.3. Classification performance on PASCAL VOC 2012

This Subsection reports the results on PASCAL Visual Object Classes Challenge (VOC) 2012 dataset [46]. VOC contains 22531 images in 20 classes for training (5717), validation (5823), and testing (10991). For this dataset, since there may be multiple labels for each image, instead of using softmax as in Subsection 4.2, sigmoid is used at the end of the network to indicate whether each class presents in an image. To demonstrate the performance of our RNN-based pooling over the existing pooling methods, networks similar as those in Subsection 4.2 are chosen where two stacks of convolutional layers are used and the number of parameters used for the convolutional layers are the same, referred to as Conv_N. Two pooling layers are used with pooling size 4×4 and 8×8, respectively, and two fully connected layers are used at the end with 256 neurons. The dataset is preprocessed into size 134×134 and random crops of size 128×128 are used as input. Considering the size of the dataset, batch size is set to 16. Other settings are the same as in Subsection 4.2, where SGD with Nesterov momentum [40] of 0.9 was used and the initial learning rate was set to 0.01. Average pooling, max pooling and the proposed RNN-based pooling are evaluated for all networks. Moreover, the VGG16 model [2] pre-trained on ImageNET is also fine-tuned on this dataset with the original max pooling and the pre-trained LSTM-based pooling (replacing the max pooling), respectively. For the fine-tuning process, the last layer of 1000 neurons with softmax function in the VGG16 model is replaced by 20 neurons with sigmoid function for the classification. The dataset is preprocessed into size 256×256 and random crops of size 224×224 as in [2] are used as input. Initial learning rate of 0.01 is used first to train the last layer with other layers fixed and then dropped by a factor of 10 to train all the layers.

The performance is shown at Table 5 in terms of mean Average Precision (mAP) [46]. For the fine-tuning process, the result on the test set (evaluated by the PASCAL VOC server) is presented while for the others, the result on the validation set is presented for simplicity. From Table 5, it can be seen that the proposed RNN-based pooling consistently improves the performance over max pooling and average pooling. Moreover, even for the fine-tuning process where the parameters are first learned based on max pooling, the proposed RNN-based pooling still improves the performance of the overall model by fine-tuning on the dataset (relatively small compared with ImageNET).

4.4. Classification performance on CIFAR-10 and CIFAR-100

In this Subsection, the widely used CIFAR-10 and CIFAR-100 datasets were used to evaluate the performance of the proposed FTN (the second architecture). The VGG16 architecture [2] was used for classification due to its popularity. Fig. 5 shows the detailed architecture. It is composed of 5 stacks of convolutional layers with a 2×2 pooling layer at the end of each stack, and 2 layers of fully connected layers in the end. Since the spatial size of the input to the fully connected layers is $1 * 1$, the fully connected layers works in the same way as convolutional layers,

Table 5: Classification result comparison on PASCAL VOC 2012 in terms of mAP (%) using different networks.

Network	Max pooling	Average pooling	Proposed pooling
Conv_32	40.9	35.1	41.5
Conv_64	43.1	34.6	44.9
VGG16 (pre-trained on ImageNET)	84.15	N/A	84.59

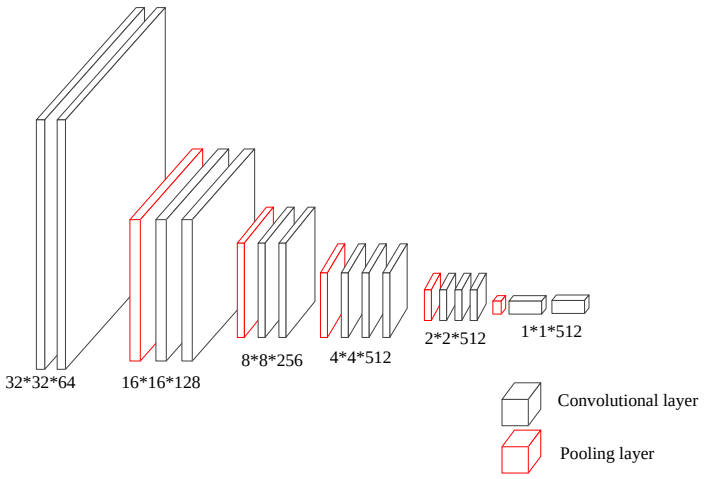


Figure 5: Illustration of the network architecture used for classification on CIFAR-10 and CIFAR-100.

and the number of the units for these two layers is reduced to 512. In the figure, the convolutional layer includes the convolution operation, batch normalization and activation function, and the size of the input to the convolutional layer and the number of neurons are also shown in the figure. A fixed 10/100 units fully connected layer together with a softmax output layer are added in the end for classification of CIFAR-10 and CIFAR-100, respectively. The leaky ReLU unit with leakiness of 0.1 was used as activation functions for both the convolutional layers and fully connected layers. Dropout was used after the pooling layers (dropping rate 30%) and fully connected layers (dropping rate 50%) to regularize the training. The preprocessing of the dataset and the training procedure are the same as in Subsection 4.2. For the proposed FTN, the pooling layers were replaced with LSTM units, one for each pooling layer. It is worth noting that one LSTM unit only introduces 12 parameters. On the contrary, one convolutional unit generally introduces $N \times N \times C_{l-1} + 1$ units where N is the filter size, C_{l-1} is the number of channels of the input to the current unit, and $+1$ indicates the bias. Compared to the large amount of parameters used in CNN, the increased number of parameters due to a LSTM unit is negligible.

The results of the FTN and the comparison to the existing methods using the similar plain CNN architectures are shown in Table 6. It can be seen that under similar training conditions (without the extreme data augmentation [51]), the proposed FTN outperforms the baseline network with max pooling and achieves better performance than other pooling methods using the similar network architecture. Although LSTM has been reported to be difficult to train in the literature, it is found that the proposed FTN converges very fast in the experiments, even faster than a CNN with traditional max pooling. The test errors of the proposed FTN and CNN v.s. the training epochs are shown in Fig. 6(a). Fig. 6(b) shows the zoomed-in curve of the testing errors of the first 20 epochs. It can be clearly seen that the proposed FTN achieved a relatively higher performance in less iterations than the CNN.

5. Conclusion

In this paper, a fully trainable network (FTN) is proposed. Compared with the traditional CNNs, the hand-crafted pooling layer is replaced with a LSTM unit in the proposed FTN. Due to the capability of a LSTM or RNN in general in modelling sequential data, the proposed learnable pooling can be trained to capture patterns of the data in the pooling regions. Specifically, we have shown that LSTM based pooling can approximate the existing pooling functions with a very high accuracy. Moreover, the proposed FTN with the LSTM based pooling

Table 6: Comparison of the proposed FTN and CNNs on CIFAR-10 and CIFAR-100 in terms of test error rate (%).

Network	CIFAR-10	CIFAR-100
DSN[47]	7.97	34.57
NIN[48]	8.81	35.68
Maxout[49]	9.38	38.57
All-CNN[9]	7.25	33.71
Highway Network[50]	7.60	32.24
ELU[42]	6.55	24.28
LSUV[43]	6.06	29.96
LSUV*[43]	5.84	N/A
LEAP[17]	7.17	29.80
Stochastic Pooling[14]	15.13	42.51
Rank based Pooling[15]	13.84	43.91
Mixed Pooling[18]	10.80	38.07
Tree Pooling[19]	6.67	33.13
Tree+Max-Avg Pooling[19]	6.05	32.37
Baseline	6.29	27.09
Proposed FTN	5.79	26.89
With extreme data augmentation[51]		
Fract. Max-pooling [51]	4.50	26.39
All-CNN[9]	4.41	N/A

LSUV* is obtained with deep residual network using maxout as activation function.
N/A represents the result is not provided in the corresponding paper.

achieves better performance in image classification over the similar networks with existing pooling methods on the CIFAR-10 and CIFAR-100 datasets.

References

References

[1] A. Krizhevsky, I. Sutskever, G. E. Hinton, Imagenet classification with deep convolutional neural networks, in: Advances in neural information processing systems, 2012, pp. 1097–1105.

[2] K. Simonyan, A. Zisserman, Very deep convolutional networks for large-scale image recognition, arXiv preprint arXiv:1409.1556.

[3] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, Z. Wojna, Rethinking the inception architecture for computer vision, arXiv preprint arXiv:1512.00567.

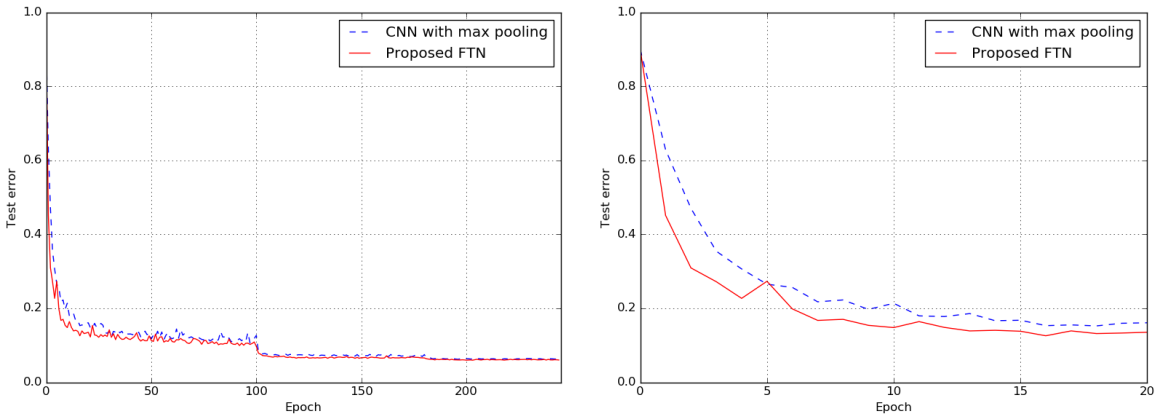
[4] M. S. Pavel, H. Schulz, S. Behnke, Object class segmentation of rgb-d video using recurrent convolutional neural networks, Neural Networks 88 (2017) 105–113.

[5] A. Karpathy, G. Toderici, S. Shetty, T. Leung, R. Sukthankar, L. Fei-Fei, Large-scale video classification with convolutional neural networks, in: Proceedings of the IEEE conference on Computer Vision and Pattern Recognition, 2014, pp. 1725–1732.

[6] K. Simonyan, A. Zisserman, Two-stream convolutional networks for action recognition in videos, in: Advances in Neural Information Processing Systems, 2014, pp. 568–576.

[7] J. Donahue, L. Anne Hendricks, S. Guadarrama, M. Rohrbach, S. Venugopalan, K. Saenko, T. Darrell, Long-term recurrent convolutional networks for visual recognition and description, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2015, pp. 2625–2634.

[8] M. Xin, H. Zhang, H. Wang, M. Sun, D. Yuan, Arch: Adaptive recurrent-convolutional hybrid networks for long-term action recognition, Neurocomputing 178 (2016) 87–102.



(a) (b)

Figure 6: Test error comparison between the proposed FTN and the traditional CNN, (a) over the entire training process, (b) over the first 20 epochs.

[9] J. T. Springenberg, A. Dosovitskiy, T. Brox, M. Riedmiller, Striving for simplicity: The all convolutional net, arXiv preprint arXiv:1412.6806.

[10] D. H. Hubel, T. N. Wiesel, Receptive fields, binocular interaction and functional architecture in the cat's visual cortex, The Journal of physiology 160 (1) (1962) 106–154.

[11] Y.-L. Boureau, N. L. Roux, F. Bach, J. Ponce, Y. LeCun, Ask the locals: multi-way local pooling for image recognition, in: Proceedings of International Conference on Computer Vision (ICCV), 2011, pp. 2651–2658.

[12] Y.-L. Boureau, J. Ponce, Y. LeCun, A theoretical analysis of feature pooling in visual recognition, in: Proceedings of the 27th international conference on machine learning (ICML-10), 2010, pp. 111–118.

[13] Z. Zhao, H. Ma, X. Chen, Protected pooling method of sparse coding in visual classification, in: Computer Vision and Graphics, Springer, 2014, pp. 680–687.

[14] M. D. Zeiler, R. Fergus, Stochastic pooling for regularization of deep convolutional neural networks, arXiv preprint arXiv:1301.3557.

[15] Z. Shi, Y. Ye, Y. Wu, Rank-based pooling for deep convolutional neural networks, Neural Networks 83 (2016) 21–31.

[16] O. Rippel, J. Snoek, R. P. Adams, Spectral representations for convolutional neural networks, in: Advances in Neural Information Processing Systems, 2015, pp. 2449–2457.

[17] M. Sun, Z. Song, X. Jiang, J. Pan, Y. Pang, Learning pooling for convolutional neural network, Neurocomputing 224 (2017) 96–104.

[18] D. Yu, H. Wang, P. Chen, Z. Wei, Mixed pooling for convolutional neural networks, in: Rough Sets and Knowledge Technology, Springer, 2014, pp. 364–375.

[19] C.-Y. Lee, P. W. Gallagher, Z. Tu, Generalizing pooling functions in convolutional neural networks: Mixed, gated, and tree, in: Proceedings of the 19th International Conference on Artificial Intelligence and Statistics, 2016, pp. 464–472.

[20] J. Feng, B. Ni, Q. Tian, S. Yan, Geometric lp-norm feature pooling for image classification, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2011, pp. 2609–2704.

[21] C. Gulcehre, K. Cho, R. Pascanu, Y. Bengio, Learned-norm pooling for deep feedforward and recurrent neural networks, in: Machine Learning and Knowledge Discovery in Databases, Springer, 2014, pp. 530–546.

[22] M. I. Jordan, Serial order: A parallel distributed processing approach, Advances in psychology 121 (1997) 471–495.

[23] S. Hochreiter, J. Schmidhuber, Long short-term memory, Neural computation 9 (8) (1997) 1735–1780.

[24] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, Y. Bengio, Learning phrase representations using rnn encoder-decoder for statistical machine translation, arXiv preprint arXiv:1406.1078.

[25] S. Li, W. Li, C. Cook, C. Zhu, Y. Gao, Independently recurrent neural network (indrnn): Building a longer and deeper rnn, arXiv preprint arXiv:1803.04831.

[26] N. Srivastava, E. Mansimov, R. Salakhutdinov, Unsupervised learning of video representations using lstms, arXiv preprint arXiv:1502.04681.

[27] W. Byeon, T. M. Breuel, F. Raue, M. Liwicki, Scene labeling with lstm recurrent neural networks, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2015, pp. 3547–3555.

[28] L. Theis, M. Bethge, Generative image modeling using spatial lstms, in: Advances in Neural Information Processing Systems, 2015, pp. 1918–1926.

[29] Y. Du, W. Wang, L. Wang, Hierarchical recurrent neural network for skeleton based action recognition, in: Proceedings of the IEEE conference on computer vision and pattern recognition, 2015, pp. 1110–1118.

[30] B. Shuai, Z. Zuo, B. Wang, G. Wang, Scene segmentation with dag-recurrent neural networks, IEEE Transactions on Pattern Analysis and Machine Intelligence.

[31] Z. Li, Y. Gan, X. Liang, Y. Yu, H. Cheng, L. Lin, Lstm-cf: Unifying context modeling and fusion with lstms for rgb-d scene labeling, in: European Conference on Computer Vision, Springer, 2016, pp. 541–557.

[32] B. Shi, X. Bai, C. Yao, An end-to-end trainable neural network for image-based sequence recognition and its application to scene text recognition, IEEE transactions on pattern analysis and machine intelligence.

[33] K. Gregor, I. Danihelka, A. Graves, D. Wierstra, Draw: A recurrent neural network for image generation, arXiv preprint arXiv:1502.04623.

[34] J. Liu, G. Wang, P. Hu, L.-Y. Duan, A. C. Kot, Global context-aware attention lstm networks for 3d action recognition, CVPR. IEEE.

[35] J. Gehring, M. Auli, D. Grangier, D. Yarats, Y. N. Dauphin, Convolutional sequence to sequence learning, arXiv preprint arXiv:1705.03122.

[36] H. T. Siegelmann, E. D. Sontag, On the computational power of neural nets, in: Proceedings of the fifth annual workshop on Computational learning theory, ACM, 1992, pp. 440–449.

[37] K. He, X. Zhang, S. Ren, J. Sun, Deep residual learning for image recognition, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2016, pp. 770–778.

[38] S. Ioffe, C. Szegedy, Batch normalization: Accelerating deep network training by reducing internal covariate shift, arXiv preprint arXiv:1502.03167.

[39] Q. V. Le, N. Jaitly, G. E. Hinton, A simple way to initialize recurrent networks of rectified linear units, arXiv preprint arXiv:1504.00941.

[40] I. Sutskever, J. Martens, G. Dahl, G. Hinton, On the importance of initialization and momentum in deep learning, in: Proceedings of the 30th international conference on machine learning (ICML-13), 2013, pp. 1139–1147.

[41] K. Hornik, Approximation capabilities of multilayer feedforward networks, Neural networks 4 (2) (1991) 251–257.

[42] D.-A. Clevert, T. Unterthiner, S. Hochreiter, Fast and accurate deep network learning by exponential linear units (elus), arXiv preprint arXiv:1511.07289.

[43] D. Mishkin, J. Matas, All you need is a good init, arXiv preprint arXiv:1511.06422.

[44] I. Sutskever, O. Vinyals, Q. V. Le, Sequence to sequence learning with neural networks, in: Advances in neural information processing systems, 2014, pp. 3104–3112.

[45] T. T. D. Team, R. Al-Rfou, G. Alain, A. Almahairi, C. Angermueller, D. Bahdanau, N. Ballas, F. Bastien, J. Bayer, A. Belikov, et al., Theano: A python framework for fast computation of mathematical expressions, arXiv preprint arXiv:1605.02688.

[46] M. Everingham, L. Van Gool, C. K. Williams, J. Winn, A. Zisserman, The pascal visual object classes (voc) challenge, International journal of computer vision 88 (2) (2010) 303–338.

[47] C.-Y. Lee, S. Xie, P. Gallagher, Z. Zhang, Z. Tu, Deeply-supervised nets, in: Proceedings of the 18th International Conference on Artificial Intelligence and Statistics, 2015, pp. 562–570.

[48] M. Lin, Q. Chen, S. Yan, Network in network, arXiv preprint arXiv:1312.4400.

[49] I. J. Goodfellow, D. Warde-Farley, M. Mirza, A. C. Courville, Y. Bengio, Maxout networks., ICML (3) 28 (2013) 1319–1327.

[50] R. K. Srivastava, K. Greff, J. Schmidhuber, Training very deep networks, in: Advances in neural information processing systems, 2015, pp. 2377–2385.

[51] B. Graham, Fractional max-pooling, arXiv preprint arXiv:1412.6071.